JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 12, 12–23 (1991)

Parallel Algorithms for Determining k-Width Connectivity in Binary Images

FRANK DEHNE*

School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6

AND

SUSANNE E. HAMBRUSCH†

Department of Computer Sciences, Purdue University, West Lafayette, Indiana 47907

In this paper we consider a new form of connectivity in binary images, called k-width connectivity. Two pixels a and b of value "1" are in the same k-width component if and only if there exists a path of width k such that a is one of the k start pixels and b is one of the k end pixels of this path. We present characterizations of the k-width components and show how to determine the k-width components of an $n \times n$ image in $O(n)$ and $O(\log^2 n)$ time on a mesh of processors and hypercube, respectively, when the image is stored with one pixel per processor. Our methods use a reduction of the k-width-connectivity problem to the 1-width-connectivity problem. A distributed, space-efficient encoding of the k-width components of small size allows us to represent the solution using $O(1)$ registers per processor. Our hypercube algorithm also implies an algorithm for the shuffle-exchange network. © 1991 Academic Press, Inc.

1. INTRODUCTION

The connected components of a binary image I partition the entries of value "1" (called the 1-pixels) into sets so that two 1-pixels are in the same set if and only if there exists a path of 1-pixels between them. Two consecutive pixels on the path are either vertically or horizontally adjacent. Determining the connected components in images is a fundamental problem in image processing [3, 8, 10, 11, 15–17]. Parallel algorithms for various architectures have been developed. When image I is of size $n \times n$ and is stored in an $n \times n$ mesh of processors with one pixel per processor, the components can be found in $O(n)$ time [1, 5, 10]. On a hypercube or shuffle-exchange network with n^2 processors, the connected components can be determined in $O(\log^2 n)$ time [2, 7]. In this paper, we formulate a stronger and more fault-tolerant form of connectivity in images, which we call k-width connectivity, and present parallel algorithms for finding the k-width components.

k-width connectivity in images captures forms of connectivity analogous to k-vertex connectivity in undirected graphs. A graph is k-vertex-connected if the removal of any $k - 1$ vertices leaves the graph connected [4]. Every image corresponds to a planar graph G in which the 1-pixels are the vertices and adjacency between two vertices corresponds to two horizontally or vertically adjacent 1-pixels. Since every such graph G contains a vertex of degree 2, G can be at most 2-vertex-connected. In order to capture stronger forms of connectivity in images, we define two 1-pixels a and b as belonging to the same *k-width component* if and only if there exists a path of width k such that a is one of the k start pixels and b is one of the k end pixels of this path. Precise definitions are given in Section 2. Figure 1 shows a path of width 3 between two pixels a and b. The image shown is not 3-width-connected since there exists, for example, no path of width 3 between pixels a and c.

The problem of determining the k-width components has a number of applications. One is in image segmentation, where an image is partitioned into coherent regions that satisfy certain requirements and relate the pixels in each region in some way [13]. Another application is the detection of connectivity in VLSI masks, where electrical connectivity between components can be maintained only by a channel whose width is never less than a value λ [9]. The image might also represent the corridors of a maze, in which case the fact that a and b are in the same k-width component implies that a robot occupying a $k \times k$ area is able to move from a to b.

In this paper we present characterizations of the k-width components and show how to determine the k-width components on a mesh of processors and a hypercube. Throughout we assume that the parallel architectures contain n^2 processors, with each processor containing $O(1)$ registers and

* Research supported in part by the Natural Sciences and Engineering Research Council of Canada.

† Research supported in part by ONR under Contracts N00014-84-K-0502 and N00014-86-K-0689 and by NSF under Grant MIP-87-15652.

0743-7315/91 $3.00

the image being stored with one pixel per processor. We develop $O(n)$ and $O(\log^2 n)$ time parallel algorithms for computing the k-width components of an image I of size $n \times n$ on a mesh and hypercube, respectively. Our methods use a reduction of the k-width-connectivity problem to the standard 1-width-connectivity problem. This reduction requires $O(k)$ and $O(\log k)$ time on a mesh and hypercube, respectively, which is asymptotically optimal. In order to represent the solution using $O(1)$ registers per processor, we use a distributed space-efficient representation of the k-width components of small size. Labeling the k-width components in a straightforward way requires $O(k)$ registers per processor (since a 1-pixel can belong to up to k k-width components). Our hypercube algorithm also implies an $O(\log^2 n)$ time algorithm for the shuffle-exchange network (with the same time and number of processors).

The remainder of this paper is organized as follows. After some basic definitions and properties are presented in Section 2, Sections 3 and 4 give characterizations of the k-width components and general strategies for determining them. The correctness of our strategies is shown in Section 5. The mesh and hypercube algorithms are presented in Section 6. Section 7 concludes our paper.

2. DEFINITIONS AND PRELIMINARIES

Consider an $n \times n$ binary image stored in a mesh or hypercube containing n^2 processors. For the mesh, we assume that the image is stored in the obvious way; i.e., the processor in row i and column j stores the pixel in the same row and column. For the hypercube, the pixels are stored with respect to the two-dimensional gray code mapping. The sequence S_n of n binary gray code numbers $gray(0), \ldots, gray(n-1)$ is defined as follows: $S_1 = (0, 1)$ and $S_n = 0*S_{n-1}, 1*(S^R_{n-1})$. Here, $0*S$ denotes the sequence of binary numbers in S each prefixed with a 0, and S^R denotes sequence S in reverse order. The two-dimensional gray code mapping is defined as $gray(i, j) = gray(i) \oplus gray(j)$, where $\oplus$ denotes the concatenation of binary numbers. The pixel in row i and column j is stored at processor $gray(i, j)$ of the hypercube. For the remainder of this paper in cases where it is obvious, we refer to the processor storing pixel x as processor x.

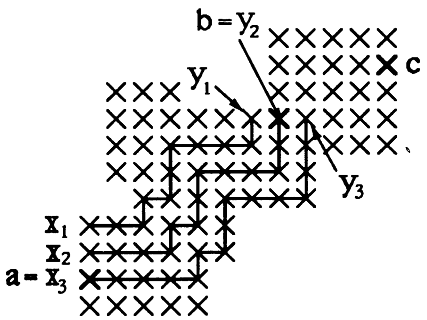

FIG. 1. A path of width 3 between 1-pixels a and b.

We now give the formal definition of k-width connectivity in images. Let x and y be two 1-pixels in image I. We assume, w.l.o.g., that no 1-pixels are located adjacent to the border of I. Let $P(x, y)$ be a path from x to y; i.e., there exist 1-pixels $x = v_0, v_1, \ldots, v_{m-1}, v_m = y$ such that v_i and v_{i+1} are horizontally or vertically adjacent. Unless stated otherwise, two pixels are called *adjacent* if they are horizontally or vertically adjacent. Consider two paths $P(x, y) = v_0, v_1, \ldots, v_m$ and $P(x', y') = w_0, w_1, \ldots, w_l$. The paths $P(x, y)$ and $P(x', y')$ are *shadow paths* if and only if (i) no pixel is contained in both paths, (ii) x and x' as well as y and y' are adjacent, and (iii) the paths are the union of a sequence $B_1, \ldots, B_l$ of 2×2 blocks of 1-pixels where B_i and B_{i+1} share exactly two adjacent pixels, $1 \leq i \leq l-1$, B_i and B_{i+2} share at most one pixel, $1 \leq i \leq l-2$, and no other two blocks share a pixel. For example, $P(x_1, y_1)$ and $P(x_2, y_2)$ from Fig. 1 are shadow paths.

Two pixels a and b are in the same k-width component if and only if there exist k mutually disjoint paths $P_i(x_i, y_i)$, $1 \leq i \leq k$, so that

- path $P_i(x_i, y_i)$ has length at least k,
- path $P_i(x_i, y_i)$ and $P_{i+1}(x_{i+1}, y_{i+1})$ are shadow paths,
- $x_1, x_2, \ldots, x_k$ (resp. $y_1, y_2, \ldots, y_k$) are on a common row or column, and
- $a = x_p$ and $b = y_r$ for some p and r.

Figures 2a and 2b show the k-width components for a given image I when $k = 2$ and $k = 5$, respectively.

A *1-block* is a subimage of I of size $k \times k$ which contains only 1-pixels. Let x be a 1-pixel of I and let $View_x$ be the $(2k-1) \times (2k-1)$ subimage of I that has pixel x in its center. Pixel x can belong to at most k^2 1-blocks and every possible 1-block containing x lies in $View_x$. The *block matrix* B_x is a boolean matrix of size $k \times k$ which records the 1-blocks pixel x belongs to. We set $B_x(i, j) = 1$ if and only if there exists a 1-block that has pixel x in row $k - i + 1$ and column $k - j + 1$ (positions are relative to the upper-left corner of the 1-block). This indexing scheme ensures that the top-left 1-block in $View_x$ corresponds to the top-left entry in the block matrix (the top-left 1-block has pixel x at position (k, k)). Figures 2c and 2d show $View_x$ and the block matrix B_x of a pixel x, respectively, when $k = 5$. For example, the "1" in the first row and fifth column of B_x indicates that there exists a 1-block in image I that has pixel x in its bottom-left corner. This 1-block is shown in Fig. 2c enclosed by dashed lines. No other 1-block contains pixel x in the bottom row and thus the first row of B_x contains no further 1's.

PROPERTY 1. *Every row (resp. column) of the block matrix B_x contains at most one contiguous sequence of 1-pixels.*

Proof. Follows from the definition of a block matrix. ∎

A 1-pixel x in image I belonging to no 1-block (i.e., every entry of B_x is "0") can obviously belong to no k-width com-

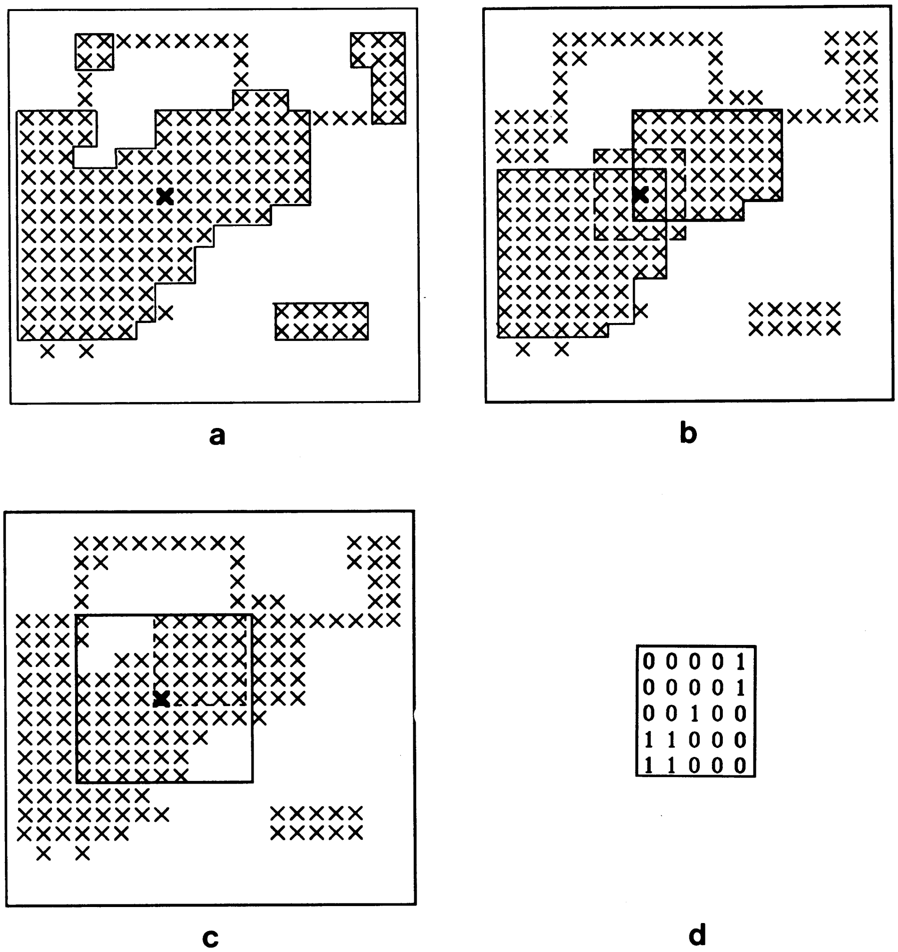

FIG. 2. Illustration of definitions: (a) the 2-width components; (b) the 5-width components; (c) *View*(x); and (d) block matrix B_x for pixel x.

ponent. Such 1-pixels are called *noise pixels.* We partition the k-width components into two types, local and global k-width components. A *local* k-width component is one whose 1-pixels can be enclosed by a rectangular region of size $(2k - 1) \times (2k - 1)$. A k-width component that is not local is called *global*. Property 2 limits the number of global k-width components a pixel x can belong to.

PROPERTY 2. *Pixel x belongs to at most two global k-width components.*

Proof. Assume that x belongs to three global k-width components. A global k-width component containing x must contain at least one 1-block corresponding to a 1-pixel on the border of B_x, as otherwise it would be contained in $View_x$. Hence, B_x contains three 1-pixels α, β, and γ belonging to different 1-width components, each on a different side on the border of B_x. W.l.o.g., let α be in row 1, β be in column 1, and γ be in row k of B_x. (The other three possibilities are handled in an analogous way.) Assume further that the column containing α is to the left of that containing γ. Let the three 1-blocks associated with these pixels be W_α, W_β, and W_γ. Pixel x is contained in all three 1-blocks. For any pixel i, let $row(i)$ (resp. $col(i)$) be the row (resp. column) containing pixel i. For the particular case considered, the bottom-right corners of W_β and W_α are contained in W_γ. This implies that the entries in B_x in $row(\beta)$ from column 1 to $col(\alpha)$ and in $col(\alpha)$ from row 1 to $row(\beta)$ are 1-pixels. Hence, α and β belong to the same 1-width component of B_x (when B_x is considered to be an image of size $k \times k$). Thus, the 1-blocks in I corresponding to α and β are in the same k-width component and the property follows. ■

The next property relates the 1-width components in block matrix B_x to the k-width components 1-pixel x can belong to in image I.

PROPERTY 3. *Let n_x be the number of* 1*-width components in block matrix B_x. Then, $n_x \leqslant k$. Furthermore, either every k-width component of image I containing pixel x corresponds to exactly one* 1*-width component of B_x (and vice versa), or one global k-width component of image I containing* *pixel x corresponds to two 1-width components of B_x and each remaining k-width component containing pixel x is a local component and corresponds to exactly one 1-width component of B_x.*

Proof. That $n_x \leqslant k$ follows immediately from the structure of the block matrix stated in Property 1. Every k-width component of image I containing pixel x induces at least one 1-width component in block matrix B_x. A local k-width component can correspond to only one 1-width component in B_x. A global k-width component corresponds to either one or two 1-width components in B_x. From the proof of Property 3 it follows that B_x cannot contain more than two 1-width components corresponding to global k-width components of image I (containing pixel x). Hence, if B_x contains two 1-width components corresponding to the same global k-width component of image I, all other 1-width components of B_x must correspond to local k-width components. On the other hand, consider the 1-pixels in image I contained in the 1-blocks associated with a 1-width component in B_x. Between any two such 1-pixels there exists a path of width k and hence they are in the same k-width component. ■

It is not difficult to design an algorithm labeling the k-width-connected components in $O(n)$ time on a mesh in which every processor has $O(k)$ registers. The main contribution of this paper lies in developing a space-efficient representation of the k-width components and an approach to compute them efficiently on a number of parallel architectures with $O(1)$ memory space per processor. A natural approach for determining the k-width components would be to "peel off" all the 1-pixels that are within distance $k/2$ from a 0-pixel and to obtain an image whose 1-width components correspond to the k-width components of image I. The straightforward application of such an approach runs into problems. Assume that k is even. Then, eliminating all 1-pixels that are within distance $k/2$ of a 0-pixel can eliminate local and global components. On the other hand, eliminating all 1-pixels that are within distance $k/2 - 1$ of a 0-pixel may not disconnect all k-width components. Similar problems arise when k is odd. Our algorithm determines the global k-width components by generating an image in which every global k-width component of I corresponds to a 1-width component of this new image. However, it is crucial that the local k-width components have been handled at this point. If we are only interested in counting the number of k-width components a simpler algorithm exists. Assume that we create from image I a new image I^* such that a pixel x in I^* is a 1-pixel if and only if x is the bottom-right pixel of a 1-block in image I. Then, it is easy to see that the number of 1-width components in image I^* is the number of local and global k-width components in image I. This method, however, does not help in deciding which components are local and global, obtaining a description of the shape of the local components, or labeling the global components in image I. Solving these problems using I^* involves essentially the same operations which we apply, in this paper, directly to I.

In the following two sections we outline our general strategy for determining the local and global k-width components. For these two sections, we assume that every pixel x has the matrices B_x and $View_x$ available. The algorithms described in Section 6 use a considerably more space-efficient representation of the information contained in the block matrices.

3. DETECTING THE LOCAL k-WIDTH COMPONENTS

From Property 3 it follows that every 1-width component in B_x represents a portion of either a local or a global k-width component. In this section we show how to detect among the 1-width components in B_x those representing local k-width components and to avoid the detection of a local k-width component by more than one processor.

We make the following convention about which processor detects which local k-width component. Processor x is in charge of detecting local k-width component C if every 1-pixel of component C is in $View_x$ and row 1 and column 1 of $View_x$ both contain one of its 1-pixels. Translated to the block matrix B_x this means that there is a 1-width component of B_x with a 1-pixel in both row 1 and column 1 of the block matrix. A processor x with such a 1-width component in its block matrix needs to determine whether the corresponding k-width component C is indeed a local k-width component (i.e., whether the respective k-width component is contained in $View_x$). Let $View_x^*$ be the $(2k + 1) \times (2k + 1)$ subimage of I that has pixel x in its center. C represents a local k-width component if no pixel adjacent to the border of $View_x^*$ is in a common k-width component with x. Efficient methods for determining this property are described in Section 6. If processor x is responsible for a local k-width component, the index of processor x is made the component number, also called the *label*, of the local k-width component.

4. DETECTING THE GLOBAL k-WIDTH COMPONENTS

In this section we outline our general strategy for determining the global k-width components. We again assume that for every pixel x the matrices B_x and $View_x$ are available. In the first step we create from image I a new image I'. We then perform a 1-width-component computation on image I', followed by a final propagation of labels to all 1-pixels in I belonging to global k-width components.

Image I' is obtained from I by changing a 1-pixel x into a 0-pixel if one of the following four conditions is satisfied:

(i) x is a noise pixel (i.e., B_x contains no 1-width component)

(ii) x belongs to a local k-width component
(iii) B_x contains two 1-width components
(iv) x is adjacent to a 1-pixel y and no 1-block contains both x and y.

Section 6 describes how to test for these conditions efficiently. Image I' contains no noise pixel, no pixel belonging to a local k-width component, no pixel belonging to two global k-width components, and no pixel of a k-width component adjacent to a pixel of another k-width component.

The following discussion (up to Lemma 4) shows that there is a one-to-one correspondence between the 1-width components of I' and the global k-width components of I; i.e., the removal of the 1-pixels from I does not eliminate a global k-width component nor does it cause one global k-width component to induce two 1-width components in I'. In order to make the necessary claims about image I', we first define the notion of s-induced and a-induced regions in a global k-width component.

Let C_i be a global k-width component and x be one of its 1-pixels. Suppose that x belongs also to another k-width component, say C_Γ. C_Γ can be a local or a global component. Let R_Γ be the largest 1-connected region shared by C_i and C_Γ which includes 1-pixel x. R_Γ is a rectangular region whose sides are of length at most $k - 1$. (Note that every border pixel of R_Γ is adjacent to 1-pixels in either C_i or C_Γ.) We say that C_Γ *s-induces* region R_Γ in C_i ("s" indicates that C_Γ and C_i share pixels). In order to define a-induced regions, suppose that pixel x is adjacent to a pixel y belonging to another k-width component, say C_Γ. Then, let R_Γ be the largest 1-connected region in C_i containing x so that every pixel in R_Γ is adjacent to a 1-pixel in C_Γ. R_Γ is a rectangle with width 1 and length at most $k - 1$. We say that C_Γ *a-induces* region R_Γ in C_i ("a" indicates that C_Γ and C_i have adjacent pixels).

The 1-pixels in R_Γ are 0-pixels in image I' since they satisfy either condition (ii) or (iii) for s-induced and condition (iv) for a-induced regions. However, conditions (iii) and (iv) may remove additional pixels from image I. When B_x contains two 1-width components and x belongs to no local k-width component, 1-pixel x may or may not belong to two global k-width components. From x's point of view, 1-pixel x does belong to two global k-width components since there exists no path of width k going through region R_Γ. If $C_i = C_\Gamma$, such a path exists by going "around" R_Γ. For the rest of this section, when we say that C_Γ induces a region in C_i we mean that C_Γ and C_i are two different k-width components from x's point of view.

We now state two properties that are used in the characterization of the interaction between induced regions. A pixel x belonging to k-width component C_i is a *corner pixel* if x is adjacent to exactly two pixels not in C_i. Note that when $k > 1$, no 1-pixel of C_i can be adjacent to three pixels not in C_i. Every s- or a-induced region R_Γ contains exactly one corner pixel of C_i, and let α_Γ be this corner pixel.

Assume that region R_Γ is s-induced by C_Γ in C_i. Let p' and p'' be the corner pixels of R_Γ, each different from α_Γ and in the same column and row as α_Γ, respectively. The position of these pixels is shown in Fig. 3a.

PROPERTY 4. *The pixel diagonally adjacent to p' (resp. p''), but not horizontally or vertically adjacent to a pixel in R_Γ cannot belong to C_i or C_Γ.*

If either of these pixels belonged to one of the components, R_Γ would not be the largest connected region. However, these pixels do not need to be 0-pixels. They can be noise pixels or belong to another k-width component.

A similar property holds when region R_Γ is a-induced. We give the statement for the case when R_Γ occupies a single row (the property for a column is similar and omitted). Let p' be the second pixel in R_Γ adjacent to only one pixel in R_Γ (with α_Γ being the first). See Fig. 3b for an illustration.

PROPERTY 5. *The pixel horizontally adjacent to α_Γ and not in R_Γ cannot belong to C_i or C_Γ. The pixel diagonally*

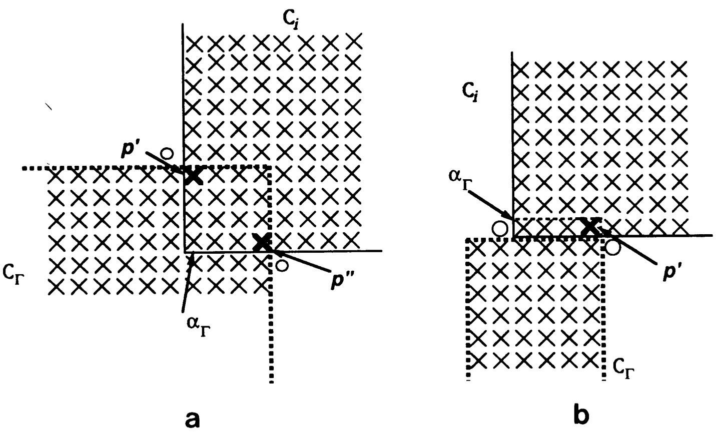

FIG. 3. Induced regions. Circled positions cannot be in either k-width component.

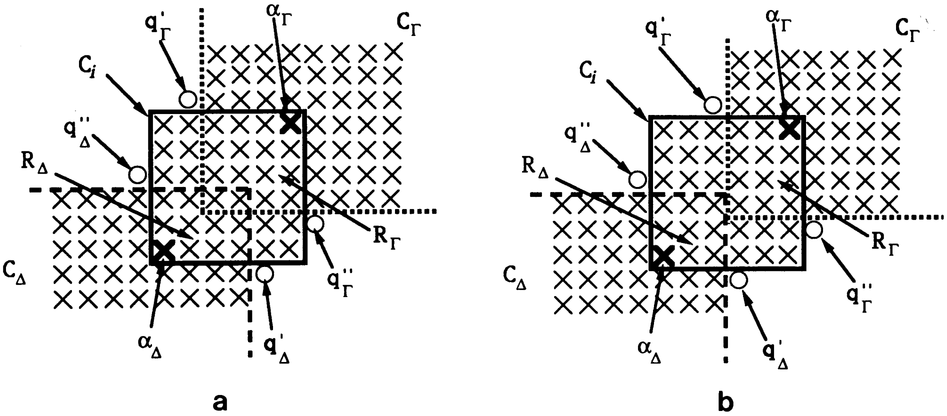

FIG. 4. Overlapping and adjacent regions with corner pixels not in the same row or column.

adjacent to p' and horizontally adjacent to a pixel in C_Γ, but not vertically adjacent to a pixel in R_Γ, cannot belong to C_i or C_Γ.

Let R_Γ and R_Δ be s-induced or a-induced rectangular regions in C_i. There are four possible relationships between R_Γ and R_Δ. One rectangle can *contain* the other one. By containment we mean that every 1-pixel of one rectangle is also in the other one and the borders of the rectangles are on different rows and columns. Obviously, no a-induced rectangle can contain another rectangle and an a-induced rectangle can only be contained in an s-induced rectangle. When R_Γ and R_Δ share pixels, but there is no containment, we say that the two rectangles *overlap.* See Fig. 4a for an example of overlapping regions. For the case when there is no 1-pixel that is in both R_Γ and R_Δ, we distinguish between disjoint and adjacent rectangles. If no 1-pixel around region R_Γ belongs to R_Δ, the two rectangles are *disjoint;* otherwise they are *adjacent.* See Figs. 4b and 5c for examples of adjacent rectangles.

Let R_i be the smallest rectangular region enclosing R_Γ and R_Δ. R_i is of size at most $(2k - 2) \times 2k - 2)$. Let α_Γ (resp. α_Δ) be the corner pixel of C_i that is in R_Γ (resp. R_Δ). Pixels α_Γ and α_Δ either are in the same row or column or are located on diagonally opposite corners of R_i. The next three lemmas characterize which relationships between two rectangles are not possible. We show that if C_i is a global k-width component, then R_Γ and R_Δ cannot overlap. R_Γ and R_Δ can be adjacent only if at least one of them is a-induced and α_Γ and α_Δ are in the same row or column.

LEMMA 1. *Let C_Γ and C_Δ be two k-width components that induce regions R_Γ and R_Δ in global k-width component C_i, respectively. If α_Γ and α_Δ are* not *in the same row or column, then R_Γ and R_Δ cannot be overlapping or adjacent.*

Proof. Assume that R_Γ and R_Δ are overlapping or adjacent with α_Γ and α_Δ on diagonally opposite corners of R_i. Then, because of Properties 4 and 5, there exist two pixels q'_Γ and q''_Γ corresponding to pixels on the border around R_i such that neither pixel belongs to C_i. See also Fig. 4. Furthermore, the position of q'_Γ and q''_Γ is such that one of them is horizontally adjacent to a pixel of R_i in $col(\alpha_\Gamma)$ and the other is vertically adjacent to a pixel of R_i in $row(\alpha_\Gamma)$. For C_Δ there exist two pixels q'_Δ and q''_Δ with the corresponding properties. Hence, every side of the border of R_i contains a pixel adjacent to a pixel that cannot belong to C_i. Let the clockwise order of these pixels be q'_Γ, q''_Γ q'_Δ, q''_Δ. There are at most $k - 2$ columns (resp. rows) between q''_Γ and q'_Δ (since each side length of R_Δ and R_Γ is of length at most $k - 1$). The same statement holds for q'_Γ and q''_Δ. This implies that the rectangular region induced by these four pixels must contain all the pixels in k-width component C_i. Thus C_i cannot be a global k-width component and the lemma follows. ■

LEMMA 2. *Let C_Γ and C_Δ be two k-width components that induce regions R_Γ and R_Δ in global k-width component C_i, respectively. If α_Γ and α_Δ are in the same row or column, then R_Γ and R_Δ cannot be overlapping.*

Proof. Observe that $\alpha_\Gamma = \alpha_\Delta$ is possible. However, it is easy to see that in this case the pixels in C_Γ and C_Δ belong to the same k-width component.

Hence assume that α_Γ and α_Δ are not identical and are, w.l.o.g., in the same row. Assume that R_Γ and R_Δ are overlapping. Let $R_{\Gamma\Delta}$ be the largest connected region of intersection between C_Γ and C_Δ that contains no pixels in R_i and contains a pixel adjacent to a pixel in R_i; see Fig. 5a. Note that $R_{\Gamma\Delta}$ cannot be empty, since each side of R_Γ (resp. R_Δ) has length at most $k - 1$. Assume w.l.o.g. that all the pixels in $R_{\Gamma\Delta}$ lie below the row containing α_Γ and α_Δ. Let q be the

[1] For two sets S_1 and S_2, $S_1 - S_2$ denotes the set containing the elements in S_1, but not in S_2.

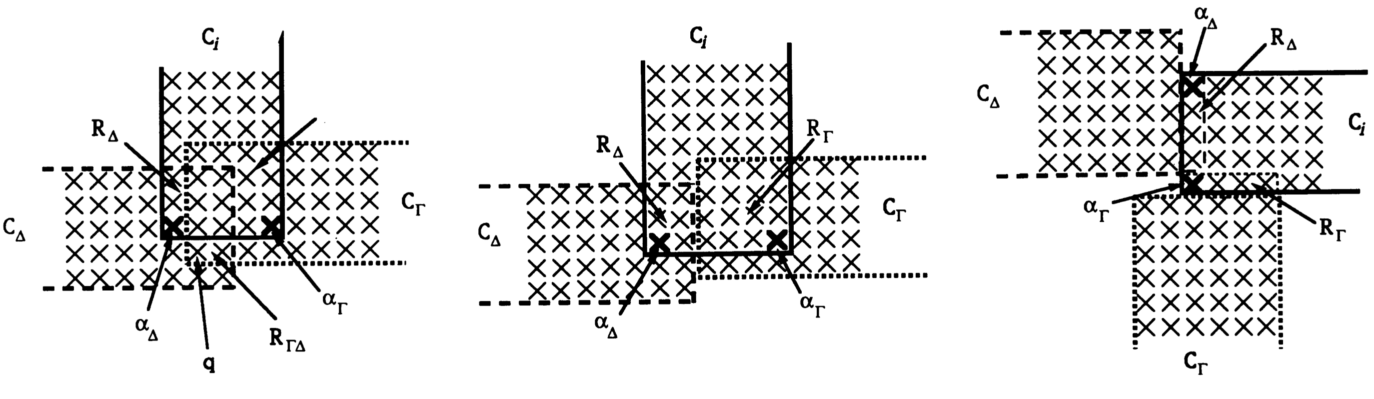

FIG. 5. Overlapping and adjacent regions with corner pixels in the same row or column.

bottom leftmost pixel in $R_{\Gamma\Delta}$. Then one of C_Γ and C_Δ, say C_Δ, contains a pixel at position $(row(q) + 1, col(q))$. If such a pixel did not exist, C_i, C_Γ, and C_Δ would belong to the same k-width component. The rectangle induced by α_Δ and q contains all 1-pixels and thus there exists a path of width k from pixels in $C_i - R_i^1$ to the pixels in $C_\Gamma - R_i$. This implies that C_i and C_Γ belong to the same k-width component and the lemma follows. ■

The final lemma addresses the possibility of adjacency when α_Γ and α_Δ are in the same row or column. An argument similar to the one used in Lemma 2 shows that, if such an adjacency occurred between two s-induced rectangles, then C_i and at least one of C_Γ and C_Δ would belong to the same k-width component. Figure 5b shows an example of such a situation.

LEMMA 3. *Let C_Γ be a k-width component that s-induces region R_Γ and let C_Δ be a k-width component that s-induces region R_Δ in global k-width component C_i. If α_Γ and α_Δ are in the same row or column, then R_Γ and R_Δ cannot be adjacent.*

Proof. Similar to the proof of Lemma 2. ■

Summarizing, we conclude that two rectangular regions R_Γ and R_Δ induced within the same global k-width component C_i cannot overlap. They can be adjacent only if both corner pixels, α_Γ and α_Δ, are in the same row or column and at least one of the regions is a-induced. Figure 5c shows a possible example of two adjacent a-induced rectangles.

We can now prove the main lemma of this section: there is a one-to-one correspondence between the 1-width components of I' and the global k-width components of I.

LEMMA 4. *Let $C_1, C_2, \ldots, C_m$ be the global k-width components of image I and let $C'_1, C'_2, \ldots, C'_l$ be the 1-width components of image I'. Then, $l = m$ and the components can be ordered so that every 1-pixel in C'_i is also a 1-pixel in C_i, $1 \leq i \leq m$.*

Proof. Let $R_{i,1}, R_{i,2}, \ldots, R_{i,\gamma_i}$ be the s- or a-induced rectangles in the global k-width component C_i. Assume that rectangles that are contained in other rectangles have been removed from this sequence. From the previous lemmas we know that no two rectangles can overlap. If $R_{i,j'}$ is adjacent to another rectangle $R_{i,j''}$, then one of them is an a-induced rectangle and both corner pixels must be in the same row or column. Furthermore, $R_{i,j'}$ can be adjacent to at most one rectangle. This is shown by using an argument similar to that used in the proof of Lemma 1. More precisely, if $R_{i,j'}$ were adjacent to two rectangles, there would exist four pixels not in C_i (see Properties 4 and 5) that would enclose C_i and thus violate the assumption that C_i is a global k-width component.

Hence, we remove from C_i either a rectangular region $R_{i,j'}$ where all the pixels around $R_{i,j'}$ and in C_i do not get deleted or a region formed by two adjacent rectangles $R_{i,j'}$ and $R_{i,j''}$. The pixels around the region formed by $R_{i,j'}$ and $R_{i,j''}$ that are in C_i are not deleted. Clearly, if regions of this structure are removed from C_i, a nonempty set of pixels remains and the pixels that remain (and which form C'_i) are 1-width-connected. Obviously, no two different sets, C'_i and C'_j, of remaining pixels are 1-width-connected. ■

After I' has been determined, we use an existing 1-width-component labeling algorithm to determine the 1-width components of I'. As a result, for each 1-width component C'_i in I', every pixel in C'_i is labeled with the same index of one arbitrary pixel x of C'_i. Note that no processor x that gave its index to a local k-width component can give its index to a global k-width component (since x is not in I'). Finally, for each component C'_i in I' its label has to be propagated to those 1-pixels of the corresponding k-width component C_i in I that are not in C'_i. Section 6 describes how this step is performed efficiently.

5. CORRECTNESS OF THE ALGORITHM

We now show that the algorithm described in the previous two sections correctly determines the k-width components of I. Let a and b be two 1-pixels of I that were assigned, by our algorithm, to the same k-width component C. If C is a local k-width component detected and recorded by processor x, then a path of width k between a and b can be obtained from B_x as follows. Let α and β be two 1-pixels in B_x such that a is in the 1-block corresponding to α and b is in the 1-block corresponding to β. The path from α to β in B_x corresponds to a sequence of 1-blocks such that two consecutive 1-blocks share a subblock of size $k \times (k-1)$. This sequence of 1-blocks represents a path of width k from a to b.

Assume now that a and b are assigned to the same global k-width component C. Let C' be the 1-width component corresponding to C in image I'. We obtain a sequence of 1-blocks corresponding to a path of width k from a path in C' and by using properties of 1-pixels in image I'. Assume first that both a and b are also 1-pixels in C' (i.e., conditions (ii)–(iv) of Section 4 do not apply to them). Let $\alpha = v_0, v_1, \ldots, v_{m-1}, v_m = b$ be a shortest path from a to b in C'. We now generate a sequence of 1-blocks that implies a path of width k from a to b.

Assume that we have generated 1-blocks $W_0, \ldots, W_{j-1}$ which represent a path of width k from v_0 to v_{i-1} that also contains $v_1, \ldots, v_{i-2}$. If v_i is a pixel in W_{j-1}, then we continue with v_{i+1} without adding another 1-block. Hence assume that v_i is not in 1-block W_{j-1}. W.l.o.g let v_{i-1} and v_i be horizontally adjacent and v_{i-1} be to the left of v_i. Let (r, c) be the position of the top-left pixel of 1-block W_{j-1}. Recall that there exists a 1-block, W', containing v_{i-1} and v_i (condition (iv) in Section 4). If there exists a 1-block containing v_{i-1} and v_i whose top-left pixel is in row r, then W_j is the 1-block that has its top-left pixel at position $(r, c+1)$, and we continue with v_{i+1}. If such a 1-block does not exist, there exists at least one 0-pixel in column $c+k$ that is adjacent to the border of W_{j-1}. W.l.o.g assume that there exists such a 0-pixel above the row containing v_i. (If there were 0-pixels above and below, then v_i could not belong to a 1-block.) Let p_0 be this 0-pixel and let $r+\delta$ be its row, $0 \leqslant \delta \leqslant k-2$ (if there exists more than one, choose the one closest to v_i). The 1-pixel p_1 of W_{j-1} in row $r+\delta+1$ and column $c+k-1$ must be the top-right corner of a 1-block W''. Otherwise, the 1-blocks W' and W_{j-1} correspond to 1-pixels in the block matrix of v_{i-1} that are in different 1-width components (in the block matrix). That is, v_{i-1} would not be a 1-pixel of I' due to condition (iii) of Section 4. Hence, we set $W_j = W'$ and $W_{j+1} = W''$ and continue with v_{i+1}.

If a and/or b are 0-pixels in image I', we obtain a path of width k as follows. If a and b belong to the same rectangular region R removed from C, such a path exists in the 1-block containing the corner pixel of C in R. If a and b belong to different rectangular regions, let a' (resp. b') be the closest pixel in C' in the same row or column as a (resp. b). Since any region deleted from C is adjacent to at most one other region, such pixels a' and b' always exist. A path of width k between a' and b' can easily be extended or modified to a path between a and b.

To complete the proof of correctness, assume that a and b are not assigned to the same k-width component. They cannot belong to the same local k-width component, since the processor detecting and recording this local k-width component would detect any path of width k between them. They also cannot belong to the same global k-width component since any path of width k between them would have resulted in a 1-width component participating in the labeling process of the respective global k-width components.

Hence, two 1-pixels a and b are assigned to the same k-width component if and only if there exists a path of width k between them.

6. PARALLEL ALGORITHM FOR MESHES AND HYPERCUBES

We now describe how to determine the k-width components of image I on a mesh and a hypercube architecture, respectively. Let us recall the steps of the strategy presented in the previous sections.

(1) For every pixel x determine the block matrix B_x and its 1-width components.
(2) Determine the local k-width components.
(3) Determine the global k-width components.

The remainder of this section is organized as follows. We first describe a technique referred to as *k-search*, which, in Section 6.2, is used for determining and recording the information contained in the block matrix. In Section 6.3 and 6.5 we show how to determine and record the local and global k-width components.

6.1. *k-Search on Meshes and Hypercubes*

Consider a row of pixels in image I. Let $x(1), \ldots, x(n)$ be these pixels and assume that each $x(j)$ has a binary value $t(j)$ associated with it. The k-search procedure consists of determining for each row of pixels and for each pixel $x(j)$ the value

$$t_k(j) = \begin{cases} \min\{r \mid t(j+r) = 1, \quad 0 \leqslant r \leqslant k\} \\ * \qquad\qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$

On a mesh, the k-search procedure can easily be executed in $O(k)$ time.

We now describe an $O(\log k)$ time implementation of k-search on the hypercube. Recall, from Section 2, that for row i, the pixel $x(j)$ is stored in processor $gray(i, j) = gray(i) \oplus gray(j)$ (where $\oplus$ denotes the concatenation of binary numbers). Let k' be the smallest power of 2 larger than or equal to k. We split each row of pixels $x(1), \ldots, x(n)$ into consecutive regions of length k' which are referred to as k'-*regions*. It is easy to see from the definition of gray codes that the processors storing the pixels of one k'-region form a subhypercube of size k'. Hence, inverse gray code conversion (cf. [6]) can be applied to each k'-region independently, in parallel. The conversion permutes the pixels in time $O(\log k)$ such that the concentrate and distribute operations of [12] can be applied. We can thus obtain, in time $O(\log k)$, for each pixel the index of the next pixel (within its k'-region) with $t(\cdot) = 1$ and the index of the leftmost pixel (within the k'-region) with $t(\cdot) = 1$. We then apply gray code conversion to each k'-region to obtain the original mapping of pixels to processors. Finally, the leftmost processor within each region communicates the index of the leftmost pixel (within the k'-region) with $t(\cdot) = 1$ to its immediate left neighbor, and each processor receiving such a value broadcasts it to all others within its k'-region. The final result is that (after $O(\log k)$ steps) each pixel has the index of the next pixel (within its k'-region) with $t(\cdot) = 1$ and the index of the leftmost pixel, within the next k'-region, with $t(\cdot) = 1$. This allows each pixel $x(j)$ to compute its value $t_k(j)$.

6.2. *Recording the Information of the Block Matrices*

In our algorithms processor x does not have block matrix B_x available, but rather a k-vertex graph, called the *block graph*. The block graph G_x contains the same information as the block matrix and it is stored in a distributed fashion so that the algorithm uses only $O(1)$ registers per processor.

Recall that the 1-pixels in a row (resp. column) of the block matrix form a contiguous sequence (Property 1). Hence, the ith column of B_x can be represented by the triple $(i, f_x(i), l_x(i))$, where $f_x(i)$ is the first row in column i containing a 1-pixel and $l_x(i)$ is the last row containing a 1-pixel, $1 \leqslant i \leqslant k$. If column i contains no 1-pixel, we set $f_x(i) = l_x(i) = 0$. The block graph $G_x = (V_x, E_x)$ for a pixel x consists of k vertices and at most $k - 1$ edges with

$$V_x = \{(i, f_x(i), l_x(i)) \mid 1 \leqslant i \leqslant k\}$$

and

$$E_x = \{((i, f_x(i), l_x(i)), (i + 1, f_x(i + 1), l_x(i + 1))) \mid$$

$$\exists z \text{ with } f_x(i) \leqslant z \leqslant l_x(i) \text{ and } f_x(i + 1) \leqslant z \leqslant l_x(i + 1)\}.$$

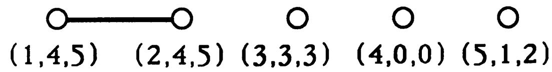

FIG. 6. Block graph G_x for the block matrix in Fig. 2.

Figure 6 shows the block graph corresponding to the block matrix shown in Fig. 2. In order to store all block graphs for all pixels, it suffices to have each pixel x store only the first node $(1, f_x(1), l_x(1))$ of its block graph G_x. The remaining $k - 1$ nodes $(2, f_x(2), l_x(2)), \ldots, (k, f_x(k), l_x(k))$ are then stored in the $k - 1$ neighbors of x, immediate to its right. To put it another way, every processor x stores only the entries $f_x(1)$ and $l_x(1)$. The ith vertex of G_x corresponds to the entries stored in processor y, where y is $i - 1$ columns to the right of processor x (and in the same row).

We now describe how to compute the f- and l-entries in time $O(k)$ and $O(\log k)$ on a mesh and hypercube, respectively. The first step is to have every 1-pixel x compute a boolean quantity br_x which is set to 1 iff x is the bottom-right corner of a 1-block (i.e., a $k \times k$ subimage of I consisting only of 1-pixels). Assume that processor x is in row r and column c in image I. In order to compute the br_x values, every processor x computes a boolean entry r_x with $r_x = 1$ iff each one of the processors at position $(r, c), (r, c - 1), \ldots, (r, c - k + 1)$ contains a 1-pixel (if one of these processors contains a 0-pixel, $r_x = 0$). After the r_x's have been determined, processor x sets $br_x = 1$ iff each processor y at position $(r, c), (r - 1, c), \ldots, (r - k + 1, c)$ has $r_y = 1$. Clearly, $br_x = 1$ if and only if x is the bottom-right corner of a 1-block. Note that the above computation reduces to two applications of the k-search procedure presented in Section 6.1.

Next, the br_x entries are used to determine $f_x(1)$ and $l_x(1)$. For each pixel x this problem reduces to searching the k pixels below x in the same column and determining the closest as well as the furthest of these with a br-value equal to 1. This computation can be performed by invoking two calls to the k-search procedure.

Summarizing, we obtain that the block graph G_x can be created on a mesh and hypercube in time $O(k)$ and $O(\log k)$, respectively, with $O(1)$ memory space per processor. It is easy to see that, using k-search, each pixel x can determine the following properties within the same time bounds:

- whether x is a noise pixel;
- whether B_x contains more than one 1-width component;
- whether the leftmost column of B_x contains a 1-pixel; let C be the 1-width component of B_x containing such a 1-pixel; and
- whether 1-width component C contains a 1-pixel belonging to row 1 of B_x.

6.3. *Determining the Local k-Width Components*

We next discuss how to detect the local k-width components. In order for a 1-pixel x, located in row r and column c, to detect and record a local k-width component, two properties need to be satisfied. First, block matrix B_x needs to contain a 1-width component that has a 1-pixel in column 1 and in row 1 of B_x. How to determine this property within the claimed time bounds follows immediately from the discussion of the preceding section. Let C be this component. Second, component C must not be k-width-connected to any pixel outside $View_x$. Component C is not k-width-connected to any pixel adjacent to the right border of $View_x$ if the following holds: let y and y' be any two pixels in row r and column $c + k - 1$ and column $c + k$, respectively. Then, if pixel y belongs to component C, the intervals $(f_y(1), l_y(1))$ and $(f_{y'}(1), l_{y'}(1))$ have an empty intersection. The conditions for not being k-width-connected to a pixel to the left of the border are similar. The conditions for a component C not being k-width-connected to a pixel adjacent to the upper border of $View_x$ are as follows. Let y be a pixel in row r and column $c + j$, $0 \leq j \leq k - 1$, belonging to component C and with $f_y(1) = 1$. Let y' be the pixel in row $r - 1$ and column $c + j$. Then, $f_{y'}(1) \neq 1$. The conditions for not being k-width-connected to a pixel adjacent to the lower border of $View_x$ are similar. The above conditions can be checked in $O(k)$ and $O(\log k)$ time, on the mesh and hypercube, respectively, by applying the k-search procedure and, for the hypercube, the concentrate and distribute operations described in [12].

Hence, we can determine which pixels are responsible for detecting a local k-width component in $O(k)$ and $O(\log k)$ time on a mesh and hypercube, respectively. As already stated, we do not explicitly label the local components (since doing so would require $O(k)$ registers per processor). If processor x detected a local k-width component, it gets marked and a convenient description of the shape of the component is obtained from the f- and l-entries. This description uses $O(k)$ registers and is stored in processor x, the $k - 1$ processors in row r immediately to the right of x, and the $k - 1$ processors in row r immediately to the left of x. Let $x = x_0, x_1, \ldots, x_{k-1}$ be the $k - 1$ processors to the right and $x_{-(k-1)}, \ldots, x_{-1}$ be the $k - 1$ processors to the left of x. For every processor x_i we determine two entries, t_i and b_i, which represent the vertical distances from row r to the top and bottom boundary pixel of component C, respectively. Any two processors in the same row such that each one of the two detected a local component are at least distance k apart. Thus, a processor can contain at most two pairs of t- and b-entries and our shape description of the local k-width components requires only $O(1)$ registers per processor. We show how to compute, for all local components, the t-values; the computation of the b-values follows from symmetry.

The f- and l-values at each processor x_i, $i \geq 0$, represent a rectangle of height $l_i - f_i + k$ and width k (except for those processors with $f_i = l_i = 0$ which represent no rectangle). The shape of a local k-width component is the union of these rectangles. For the mesh, the t-values can be computed in time $O(k)$ simply by shifting the f- and l-values k positions to the left. In the remainder of this section we describe an $O(\log k)$ time solution for the hypercube. We start by defining the partial prefix operation which, together with simple routing operations, is the main ingredient for determining the t-values.

Assume that every processor p_i in a k'-dimensional hypercube contains a value a_i and two processors, p_s and p_t with $s \leq t$, are marked. In the *left partial prefix* every processor p_j with $s \leq j \leq t$ determines $\max\{a_s, a_{s+1}, \ldots, a_j\}$. In the *right partial prefix* every processor p_j determines the entry $\max\{a_j, a_{j+1}, \ldots, a_t\}$. Straightforward changes to known parallel prefix algorithms allow us to determine the left and right partial prefix in $O(\log k')$ time on a hypercube of dimension k'.

For every processor x_i, $i \geq 0$, with $f_i > 0$, let $t'_i = k - f_i$ and 0 otherwise. For $-(k - 1) \leq j \leq 0$, we have $t_j = \max\{t'_0, t'_1, \ldots, t'_{k+j-1}\}$. For $1 \leq j \leq k - 1$, we have $t_j = \max\{t'_j, t'_{j+1}, \ldots, t'_{k-1}\}$. In order to compute the t-values for $-(k - 1) \leq j \leq 0$, we first compute a left partial prefix on the t'-values with processors x_0 and x_{k-1} being marked and send the entry computed by processor x_i to processor x_{i-k+1}. We then compute the t-values for $1 \leq j \leq k - 1$. This is done simply by performing a right partial prefix on the t'-values with processors x_0 and x_{k-1} being marked.

We conclude the computation of the shape description by sketching how to perform the partial prefix computations in $O(\log k)$ time. Let k' denote, again, the smallest power of 2 larger than or equal to k, and view each row of processors as being split into a sequence of blocks of length k'. A region on which we need to perform a partial prefix operation can lie entirely within a block or it can be split over two blocks. Three partial prefix operations on subhypercubes of dimension k' can easily produce the necessary values. In the first we perform a partial prefix on all regions that lie entirely within a block. The next two handle the regions that are split: the second parallel prefix works with the beginning segments of the region and the third with the ending segments of the regions. It is straightforward to combine the results of the second and third partial prefix computations. Hence, the description of the shape in terms of the t- and b-entries can be generated in $O(\log k)$ time.

6.4. *Determining the Auxiliary Image I'*

As stated earlier, the global k-width components are determined by computing the 1-width components of an auxiliary image I' obtained from I by changing a 1-pixel x into a 0-pixel if one of four conditions is satisfied. In this section, we describe how to obtain image I' in time $O(k)$ and $O(\log k)$ on the mesh and hypercube, respectively.

First, we need to change all noise pixels to 0-pixels. Following Section 6.2, this is immediate.

Second, we require that all pixels belonging to any local components be marked (and subsequently changed to 0-pixels). If processor x detected a local k-width component, then processor x, the $k - 1$ processors to the right of x, and $k - 1$ processors to the left of x each contain a t-value and a b-value describing the shape of this component (see Section 6.3). The marking can be done by having each processor x_i that stores values t_i and b_i broadcast a marker to the t_i and b_i pixels above and below x_i (and in the same column), respectively. Each such broadcast is restricted to a neighborhood of k pixels and can therefore be executed on a mesh in time $O(k)$. On a hypercube, each such broadcast can be executed in time $O(\log k)$ using techniques already described.

Next, we need to delete all pixels whose block matrix contains two or more 1-width components. Following Section 6.2, this is immediate.

Finally, we need to determine those pixels x that are adjacent to a 1-pixel y but with no 1-block containing both x and y. We now describe how to use the block graphs to determine whether two adjacent 1-pixels x and y are not contained in a common 1-block. Assume that x is to the left of y. When no 1-block contains both x and y, G_x consists of one connected component formed by vertex $(1, f_x(1), l_x(1))$ and G_y consists of one connected component formed by vertex $(k, f_y(k), l_y(k))$ (all other values in G_x and G_y are zeroes). Assume that x is above y. Pixels x and y are in no common 1-block if G_x and G_y contain one connected component each, the vertices of G_x have the values $f_x(j) = l_x(j) = 1$, and the vertices of G_y have the values $f_y(j) = l_y(j) = k$. If x is to the right of or below y, similar arguments hold. All of these tests can be implemented by a k-search procedure.

Summarizing, we obtain that the auxiliary image I' can be computed in $O(k)$ and $O(\log k)$ time on a mesh and hypercube, respectively.

6.5. *Determining the Global k-Width Components*

Once the auxiliary image I' has been determined, an algorithm for determining the 1-width components of I' is applied [1, 5, 10, 2, 7]. This requires time $O(n)$ and $O(\log^2 n)$ on the mesh and hypercube, respectively. As shown in Section 4, the 1-width components of I' correspond exactly to the global k-width components of I. The final step consists of propagating the labels to the 1-pixels of global k-width components which are not in I'. Note that each such pixel belongs to at most two global k-width components (see Section 2) while the pixels in I' belong to exactly one global k-width component. Furthermore, each global k-width component is the union of the 1-blocks indicated in the block matrices of the pixels belonging to the respective 1-width component in I'. The propagation of the labels can therefore be accomplished in essentially the same way as the computation of the shape of the local components and the marking of the pixels that belong to any local component, described in the previous two sections (requiring time $O(k)$ and $O(\log k)$ time on a mesh and hypercube, respectively).

7. CONCLUSION

In this paper we have presented $O(n)$ and $O(\log^2 n)$ time parallel algorithms for computing the local and global k-width components of an image I of size $n \times n$ on a mesh and hypercube, respectively, requiring $n \times n$ processors and $O(1)$ memory space per processor. The hypercube algorithm immediately implies a shuffle-exchange network algorithm with the same time complexity.

The presented mesh algorithm is asymptotically optimal. It is worthwhile to note that, besides the time for determining the 1-width components of the auxiliary image I', our methods requires only time $O(k)$ and $O(\log k)$ time on a mesh and hypercube, respectively. Hence, our algorithms can also be viewed as a $O(k)$ and $O(\log k)$ time, respectively, reduction of k-width connectivity to 1-width connectivity. In that sense, our reduction algorithm is asymptotically optimal for both the mesh and the hypercube architecture.

ACKNOWLEDGMENTS

We thank Greg Frederickson and Mike Atallah for helpful comments and suggestions.

REFERENCES

1. Cypher, R., Sanz, J., and Snyder, L. Algorithms for image component labeling on SIMD mesh connected computers. *IEEE Trans. Comput.* **39** (1990), 276–281.
2. Cypher, R., Sanz, J., and Snyder, L. Hypercube and shuffle-exchange algorithms for image component labeling. *J. Algorithms.* **10** (1989), 140–150.
3. Dyer, C., and Rosenfeld, A. Parallel image processing by memory-augmented cellular automata. *IEEE Pami* **3** (1981), 29–41.
4. Harary, F. *Graph Theory.* Addison–Wesley, Reading, MA, 1972.
5. Hambrusch, S. E., and TeWinkel, L. A study of connected component algorithms on the MPP. *Proc. 3rd International Conference on Supercomputing,* 1988, pp. 477–483.
6. Johnsson, S. L. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distrib. Comput.* **5** (1987), 133–172.
7. Lim, W., Agrawal, A., and Nekludova, L. A fast parallel algorithm for labeling connected components in image arrays. Tech. Rep. NA86-2, Thinking Machines Corp., 1986.
8. Levialdi, S. On shrinking binary picture patterns. *Comm. ACM* **15** (1972), 7–10.
9. Mead, C. A., and Conway, L. A. *Introduction to VLSI Systems.* Addison–Wesley, Reading, MA, 1980.

10. Miller, R., and Stout, Q. *Parallel Algorithms for Regular Architectures.* MIT Press, Cambridge, MA, in press.
11. Nassimi, D., and Sahni, S. Finding connected components and connected ones on a mesh-connected parallel computer. *SIAM J. Comput.* (1980), 744–757.
12. Nassimi, D., and Sahni, S. Data broadcasting in SIMD computers. *IEEE Trans. Comput.* **30** (1981), 101–107.
13. Pavlidis, T. *Algorithms for Graphics and Image Processing.* Computer Sci. Press, 1982.
14. Preparata, F., and Shamos, M. *Computational Geometry—An Introduction.* Springer-Verlag, New York/Heidelberg, 1985.
15. Rosenfeld, A. Connectivity in digital pictures. *J. Assoc. Comput. Mach.* **17** (1970), 146–160.
16. Rosenfeld, A., and Kak, A. *Digital Picture Processing.* Academic Press, San Diego, CA.
17. Veillon, F., One pass computation of morphological and geometrical properties of objects in digital pictures. *Signal Process.* **1** (1979), 175–189.

Received July 24, 1990; accepted November 21, 1990

FRANK DEHNE was born in Hannover (West Germany) in 1960. He received his M.C.S. degree (Dipl. Inform.) from the Technical University of Aachen (West Germany) in 1983 and his Ph.D. (Dr. rer. nat.) from the University of Würzburg (West Germany) in 1986. He is currently an associate professor of computer science at Carleton University in Ottawa. His research interests include computational geometry, data structures, and parallel and VLSI algorithms. Since 1989, Dr. Dehne has served as Managing Editor for *Information Processing Letters.*

SUSANNE E. HAMBRUSCH received her Diplom Ingenieur in computer science from the Technical University of Vienna (Austria) in 1977 and her Ph.D. degree in computer science from the Pennsylvania State University (University Park, PA) in 1982. In 1982 she joined the Computer Science Department at Purdue University (West Lafayette, IN), where she is currently an associate professor. Her research interests include parallel computation, analysis of algorithms, image processing, combinatorics, and graph theory. Dr. Hambrusch is a member of the Association for Computing Machinery, the IEEE Computer Society, and the Society for Industrial and Applied Mathematics.